TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika

Studijní obor: 1802R022 – Informatika a logistika

Využití SoftPLC při řízení fyzikálních modelů

Use of SoftPLC for controlling of physical models

Bakalářská práce

Autor: **Milan Kužela**

Vedoucí práce: Ing. Miloš Hernych

Konzultant: doc. Ing. Jiřina Královcová, Ph.D.

V Liberci 25. 12. 2011

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Akademický rok: **2011/12**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Milan Kužela**
Osobní číslo:	**M06000229**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Informatika a logistika**
Název tématu:	**Využití SoftPLC při řízení fyzikálních modelů**
Zadávající katedra:	**Ústav mechatroniky a tehnické informatiky**

Zásady pro vypracování:

1. Seznamte se s komunikačními protokoly PLC Tecomat a možností sdílení paměti u programu Tecomat SoftPLC.

2. Navrhněte způsob řízení fyzikálních modelů v učebně TK3 sdílením paměti programu Tecomat SoftPLC s využitím stávajícího HW vybavení.

3. Návrh realizujte, vytvořte pedagogickou dokumentaci pro jednotlivé úlohy a popište praktické zkušenosti s provozem.

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum .....

Podpis .....

Poděkování

Chtěl bych na tomto místě poděkovat všem lidem, kteří nějakým způsobem napomohli vzniku této práce. Jmenovitě bych rád vyjádřil svůj dík vedoucímu mé práce, panu Ing. Miloši Hernychovi za cenné rady, připomínky a nesmírnou trpělivost, dále panu Ing. Pavlu Schusterovi ze společnosti Teco a.s. za poskytnutí materiálů týkajících se modulu sdílené paměti SoftPLC a v neposlední řadě mé poděkování patří i celé Fakultě mechatroniky a mezioborových inženýrských studií. Zvláštní poděkování bych rád věnoval své rodině za jejich obrovskou podporu.

Abstrakt

Cílem této práce je navrhnout a vytvořit aplikaci pro přenos dat mezi softwarovým emulátorem SoftPLC for Windows a jeho reálnou obdobou, PLC automatem řady Tecomat TC700, který má za úkol ovládat reálné fyzikální úlohy k nimž je připojen. K výměně dat mezi emulátorem a touto aplikací bylo využito modulu sdílené paměti situovaného v dynamické knihovně ShmDll.dll, která je součástí instalace SoftPLC. K vlastnímu přenosu dat mezi pracovní stanicí s nainstalovaným SoftPLC a reálným automatem bylo využito sériového rozhraní RS-232 a protokolu EPSNET.

Tento program je určen pro výhradně platformu Windows a k jeho vytvoření bylo použito vývojového prostředí Microsoft Visual Studio 2010 a jazyka C++ se snahou o co největší užívání funkcí Windows API.

Klíčová slova: PLC, SoftPLC, RS-232, sériová komunikace, modul sdílené paměti

Abstract

The objective of this thesis was to design and implement an application for data transfer between software emulator SoftPLC for Windows and its real analogy, PLC controller of Tecomat TC700 series, whose job is to control the real physical tasks it is attached to. For data exchange between the emulator and this application a shared memory module situated in the ShmDll.dll library of the SoftPLC installation was used. For actual data transfer between the workstation with SoftPLC installed and real controller, a RS-232 serial interface and EPSNET protocol was used.

This program is designed exclusively for Windows operating system and was created in development environment of Microsoft Visual Studio 2010 and C++ programming language with the aspiration to use Windows API functions as much as possible.

Keywords: PLC, SoftPLC, RS-232, serial communication, shared memory module

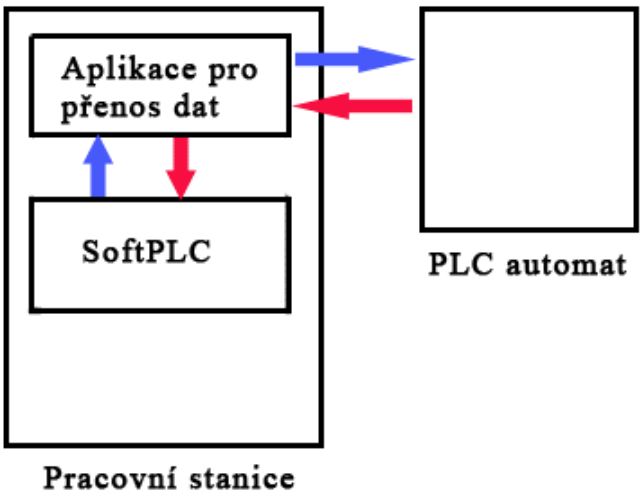
1 Úvod

SoftPLC for Windows (dále jen SoftPLC) od společnosti Teco a.s. představuje softwarový emulátor reálného PLC automatu se všemi jeho funkcemi a možnostmi. To nám dává možnost využít ho jako řídícího prvku v systému připomínající server - klient architekturu. Stanice s nainstalovaným a spuštěným SoftPLC bude tvořit centrální prvek systému (server) a zajišťovat všechny řídící a systémové funkce. K této stanici se pomocí sériové linky RS-232 připojí podřízené moduly tvořené reálnými PLC automaty (klienti). Zjednodušeně lze říci, že všechna činnost takového systému bude pod kontrolou výše zmíněné nadřazené stanice.

Hlavním pozitivem takového modelu je, že tento nadřazený centrální prvek může sloužit jako operátorské středisko kombinující všechny nutné funkce potřebné pro chod systému a jeho snadnou údržbu (například vizualizace systému, diagnostika, správa dat). Avšak nic není zadarmo, největší přednost tohoto řešení se stává i jeho největší slabinou. Stejně jako všechny ostatní systémy s centrálním řízením, i tento však trpí velmi závažnou nevýhodou. Dojde li k výpadku centrálního prvku nebo k přerušení komunikačních kanálu s podřízenými stanicemi (např. v důsledku rušení či fyzického poškození) , celý systém se stává neovladatelným.

Proto se v praxi stále setkáme spíše s decentralizovaným systémem, kde řízení zajišťují jednotlivé klasické PLC automaty. Využití SoftPLC lze očekávat spíše v případech, kdy by byl systém proti výpadku centrálního prvku zabezpečen nějakým sekundárním systém nebo v případech, kdy pozitiva tohoto řešení jasně převáží nad jeho negativy - tedy zejména v případech, kdy je potřeba řídit složité operace a zpracovávat velká množství dat.

Cílem této práce je vytvořit aplikaci, která bude výše naznačeným způsobem umožňovat řídit systém pomocí SoftPLC. Zjednodušeně řečeno, tato aplikace bude představovat jakési rozhraní pro komunikaci mezi emulátorem a automatem. Předpokládejme, že je řízený fyzikální model napojen na vstupy automatu. Je zřejmé, že budeme potřebovat data ze vstupů automatu pravidelně předávat SoftPLC, kde budou zpracovávány uživatelským programem, a na jejich základě naopak předávat jiná data zpět automatu. Problém je zachycen na obrázku č.1.



obrázek 1: Blokové schéma komunikace mezi PLC a SoftPLC

2 Analýza problému

Abychom mohli navrhnout program, který bude schopný plnit výše zmíněnou funkci, je třeba si nejprve ujasnit, jak bude samotný program fungovat, jaké dílčí úlohy bude třeba vyřešit a jaké jsou možnosti jejich řešení.

2.1 Volba jazyka

Pro vytvoření programu jsem zvolil vývojové prostředí Microsoft Visual Studio a programovací jazyk C++. Vedlo mě k tomu několik důvodů. Kromě spojení rychlosti kódu s možností objektového přístupu, byl jedním z nich i fakt, že se ve svém programu potřebuji napojit na modul sdílené paměti SoftPLC, což v praxi znamenalo přidat do mého projektu soubory poskytnuté společností Teco a.s., přičemž byly tyto soubory psány právě v C++. Ačkoliv by bylo možné tyto soubory převést do jiného jazyka, považoval jsem za lepší řešení psát svůj program také v tomto jazyce. Byla tím zajištěna jejich kompatibilita s mým programem a navíc odpadla práce s jejich případnou konverzí do jiného jazyka.

Dalším důvodem bylo, že se ve svém programu snažím využívat funkcí Windows API. Ty jsou v z C++ relativně snadno dostupné, navíc dokumentace a příklady jejich používání jsou na portálu MSDN (Microsoft Developer Network) obvykle uváděny právě v tomto jazyce.

2.2 Komunikace se SoftPLC

Je zřejmé, že bylo nutné vyřešit způsob, jakým bude můj program komunikovat se SoftPLC. V zásadě bylo možné využít jednu ze dvou možností, využít null-modem emulátor nebo modul sdílené paměti SoftPLC.

2.2.1 Null-modem emulátor

Null-modem emulátor je software (jako příklad na trhu dostupných aplikací tohoto typu je možné uvést například emulátor *com0com* nebo *Virtual null-modem*), který na daném počítači dokáže vytvořit dvojici vzájemně propojených virtuálních sériových portů COM s virtuálně překříženými vodiči Rx a Tx. Například vytvoříme-li virtuální pár COM2-COM3, tak data odeslaná na COM2 budou doručena na COM3 a naopak. Toho lze využít i pro komunikaci se SoftPLC. Celá komunikace v podstatě vychází z principu komunikace po sériové lince (viz kapitoly 6 a 7), SoftPLC obsadí dejme tomu COM2, naše aplikace COM3. Pak zbývá jen odeslat požadavek na čtení či zápis do zápisníku SoftPLC a počkat na odpověď.

Nevýhoda tohoto řešení je však zřejmá – je nutné využívat další externí aplikaci pro emulování virtuálních portů a to je nepraktické hned z několika důvodů. Uživatel nemusí být seznámen s tím, jak ji správně nakonfigurovat a používat, jednak jakákoliv další aplikace zbytečně zatěžuje daný počítač.

2.2.2 Modul sdílené paměti

Modul sdílené paměti se nachází v dynamické knihovně ShmDll.dll, která je součástí instalace SoftPLC a mohou ji využívat aplikace třetích stran pro přímý přístup do pamětového prostoru (zápisníku). Zjednodušeně řečeno se v této knihovně nacházejí data pro všechny instance SoftPLC a dále funkce pro přístup k zápisníku a manipulaci s daty. Oproti předchozí variantě má tato metoda tu výhodu, že není potřeba instalovat a konfigurovat žádný další program. Z toho důvodu budu tuto meto vzájemné komunikace mezi svým programem a SoftPLC využívat v mé práci. Vzhledem k tomu, že se tato knihovna může nacházet na různých místech podle toho, kam uživatel SodtPLC nainstaluje, bude nutné v konfiguračním souboru mé aplikace specifikovat umístění této knihovny.

2.3 Komunikace s PLC automatem

Pro výměnu dat s PLC automatem bude využito propojení z bodu do bodu pomocí sériového rozhraní RS-232. Zároveň automat poběží v režimu PC, což znamená, že se bude chovat jako podřízený účastník, přičemž komunikace bude iniciována účastníkem nadřízeným a bude probíhat na principu dotaz – odpověď. Přenos dat bude probíhat prostřednictvím protokolu ESPNET, který bude dále podrobněji popsán v kapitole 7.

2.4 Konfigurace programu

Vzhledem k tomu, že můj program je zamýšlen pro běh „na pozadí“ operačního systému, tak jsem potřeboval vyřešit problém, jak by uživatel mohl předat programu parametry nutné k jeho chodu. To mě přivedlo k myšlence, aby si program nutná vstupní data načetl z nějakého externího umístění. Otázkou zůstalo jak tento problém implementovat, nabízí se několik řešení, každé má své klady i zápory.

a) pomocí obyčejného souboru

Toto řešení by vyžadovalo vytvořit nějaký binární soubor s přesně vymezenou strukturou dat. Problém tohoto řešení je poměrně velká programátorská náročnost, přičemž bychom nutně museli uživateli poskytnout nějaký nástroj, který by daný datový soubor dokázal vytvořit a upravit.

b) pomocí registrů Windows

Ačkoliv je používání registrů Windows značně rozšířené, rozhodl jsem se mu vyhnout. Vedlo mě k tomu několik důvodů. Prvním z nich je, že při odinstalaci mého programu by také bylo nutné (či spíše z programátorského hlediska etické) odebrat z registrů vše, co tam uživatel zapíše, což by znamenalo řešit další problém navíc. Faktem také zůstává, že velká část problémů, kterým uživatel Widows často čelí, je způsobená právě chybou v registrech. Ačkoliv systém Windows obsahuje nástroj *regedit*, jen málokterý uživatel s ním kdy přišel do styku a dokázal by ho použít k nastavení výchozích parametrů aplikace.

c) pomocí databáze

Toto řešení, ač na první pohled elegantní, se v mém případě nevyplatí. Sice umožňuje přehledně uchovávat jasně strukturovaná data, ale zároveň by bylo nutné mít k dispozici

databázovou aplikaci vyřešit problém jak do ní data zapisovat naopak z ní data číst. Mělo by to snad smysl, kdybychom potřebovali uchovávat opravdu velká množství dat, avšak to v našem případě neplatí, proto jsem tuto možnost řešení zavrhl.

d) pomocí INI souboru

Konfigurační .ini soubory jsou malé strukturované textové soubory, do nichž je možné pomocí prostého textu zapsat data pro počáteční konfiguraci programu. Z programátorského hlediska je jejich využití velmi výhodné, protože programovací jazyky zpravidla obsahují funkce, které nám umožní relativně jednoduše do nich strukturovaně data zapisovat i číst. Vzhledem k tomu, že se jedná o prostý textový soubor, může si ho uživatel prohlédnout a upravit aniž by k tomu potřeboval nějaký speciální program (postačí k tomu například *Poznámkový blok*, který je běžně součástí Windows). Je zde však nutné, aby uživatel znal význam jednotlivých položek a možných hodnot. Toho můžeme docílit například pomocí komentářů v daném ini souboru, ale abychom uživateli usnadnili život, můžeme vytvořit i jednoduchou formulářovou aplikaci, která bude sloužit k vytvoření a úpravě .ini souboru.

Poměrně velkou výhodu představuje i fakt, že při odinstalaci programu není třeba řešit žádný další problém jako tomu bylo třeba u registrů, ini soubor stačí prostě smazat. Kvůli výše popsaným výhodám jsem se rozhodl využít toto řešení v mém programu.

2.5 Běh na pozadí a spuštění při startu systému

Vzhledem k tomu, že program veškerou svoji činnost provádí mimo zrak uživatele, není tedy nutné aby okno s běžícím programem zabíralo místo na obrazovce. Z toho důvodu jsem se rozhodl, že programu umožním „minimalizovat se“ do oznamovací oblasti (system traye). V praxi to znamená, že se při stlačení tlačítka pro minimalizaci hlavní okno programu skryje a v oznamovací oblasti se objeví ikonka programu. Dvojklikem na tuto ikonu pak bude možné hlavní okno programu znovu obnovit.

Další otázkou bylo, zda umožnit programu automaticky se spouštět při startu systému. Ačkoliv by to možná bylo pohodlnější pro uživatele, který by se tak nemusel téměř o nic kromě připojení automatu a spuštění SoftPLC starat, nakonec jsem se rozhodl této možnosti nevyužít. Vedl mě k tomu zejména fakt, že ne vždy bude uživatel používat počítač k práci se SoftPLC a automatem. V těchto případech je zcela zbytečné,

aby se program zaváděl automaticky do paměti a zabíral tak systémové prostředky. Bude-li přesto uživatel chtít spouštět program automaticky při startu systému, stále může například zástupce programu umístit do nabídky *Start/Programy/Po Spuštění* (jinou možností by bylo editovat registry Windows pomocí nástroje *regedit* a do větve *HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run* zapsat název aplikace a cestu k spouštěnému programu).

3 Platforma Windows z programátorského hlediska

Než se dostaneme k detailnímu popisu řešení problémů týkajících se této aplikace, tak by jistě nebylo na škodu seznámit se s některými principy fungování systému Windows a s tím, jak jsou vlastně programy běžící pod tímto operačním systémem řízeny a ovládány.

3.1 Architektura systému Windows

Operační systém Windows pracuje na principu systému server-klient. Aplikace běžící na tomto systému nemají možnost přistupovat přímo k hardwaru. Na operační systém můžeme nahlížet jako na mezivrstvu mezi uživatelskými programy a hardwarovým vybavením počítače. Veškeré dění tedy prochází přes Windows, což je staví do role serveru, zatímco jednotlivé aplikace představují klienty.

Operační systém musí dokázat řešit velké množství komplexních problémů. Z programátorského hlediska je pro nás nejzajímavější vědět, jakým způsobem je realizována komunikace mezi OS a uživatelským programem a jak se řeší běh více aplikací najednou.

Pokud některá aplikace potřebuje vyřídit nějakou operaci, nemůže tak učinit sama, jelikož nemá přístup k hardwaru, ale musí požádat operační systém a ten se jí pokusí vyhovět.

Aby bylo možné takový požadavek obsloužit, musí existovat nějaký mechanismus vzájemné komunikace mezi OS a aplikacemi. Není divu, že takový mechanismus skutečně existuje - komunikace se na Windows uskutečňuje prostřednictvím tzv. zpráv a je zřejmé, že komunikace probíhá oběma směry.

Pomocí zpráv také operační systém předává aplikacím informace o různých událostech, které se jich týkají (například stisk tlačítka, překreslení obrazovky) a je poté na aplikaci

aby zachycenou zprávu zpracovala podle potřeby. Událostmi řízené programování je tedy do značné míry právě o zasílání a přijímání zpráv a reagování na ně.

3.2 Procesy, vlákna, multitasking, multithreading

Pod pojmem proces chápeme vykonávaný program. Můžeme na něj nahlížet jako na soubor informací, které má operační systém k dispozici, například kód programu, proměnné v paměti, data na disku a další systémové informace. Každý program, respektive proces, má alespoň jedno vlákno, může jich tedy být i více. Na vlákno můžeme nahlížet jako sled instrukcí, které jsou zpracovávány procesorem.

Převážná většina operačních systémů, se kterými se běžně setkáváme, umožňuje tzv. multitasking (výjimku tvoří například dnes již velmi zřídka používaný DOS). Multitasking znamená, že operační systém podporuje současný běh více procesů najednou.

Vzhledem k tomu, že běžné osobní počítače obvykle mívají jen jeden procesor, není možné, aby všechny programy běžely současně. Situace se řeší tak, že procesor střídavě vykonává kód jednotlivých programů (dochází k tzv. *změně kontextu*) a díky tomu se vytváří iluze jejich současného běhu. Je záležitostí operačního systému, aby určil, který program bude zpracováván procesorem. Obvykle se tak děje na základě priority procesu.

Zároveň je potřeba určit, na jak dlouho bude mít program procesor k dispozici. Tyto časové úseky nazýváme *časovými kvanty*, ta jsou obvykle dlouhá několik desítek milisekund, avšak není to pravidlem. Pokud OS podporuje tzv. preemptivní multitasking, je délka časových kvant plně v režii operačního systému, u nepreemptivního multitaskingu byla nutná aktivní spolupráce aplikace (aplikace se sama musela vzdát procesoru, což v případě chybně naprogramované aplikace často vedlo k zamrznutí celého systému). Samotné plánování délky časových kvant vyžaduje jistý kompromis mezi rychlou odezvou programů (krátká časová kvanta) a efektivitou (delší časová kvanta), jelikož příliš časté přepínání programů znamená velký podíl režijních úkonů (např. záloha registrů).

Má-li daná aplikace více vláken, mluvíme o multithreadingové nebo vícevláknové aplikaci. V praxi to znamená, že se program dokáže větvit do několika vláken, která mohou běžet paralelně. V rámci své aplikace pak tato vlákna sdílejí i systémové zdroje a mají tak například přístup ke globálním proměnným programu.

Vícevláknové aplikace nacházejí největší uplatnění v případě, kdy potřebujeme vykonávat časově náročné operace (složité výpočty, práce se soubory apod.). Obsluha takové operace se svěří samostatnému vláknu zatímco jiná vlákna mohou dál vykonávat další činnost aniž by byla jakkoliv zpomalována.

3.3 Zprávy

Jak již bylo výše naznačeno, veškeré objekty v systému Windows mezi sebou komunikují prostřednictvím zpráv. Za jejich generování a předávání je zodpovědný právě operační systém, který jejich prostřednictvím informuje aplikace o vzniku nejrůznějších událostí, na které by mohly aplikace chtít reagovat. To jestli na ně aplikace opravdu zareaguje, je však čistě na ni. Podívejme se nyní, jak je taková zpráva v systému Windows realizována. Je to v podstatě datová struktura, obsahující následující položky:

a) handle okna – jedná se o unikátní identifikátor, který určuje, jakému objektu je zpráva určena
b) typ zprávy – číselný kód určující význam zprávy, např. zpráva WM_TIMER je reprezentována číslem 0x0113
c) parametr wParam – jedná se o dvoubytové číslo upřesňující význam zprávy, v případě zprávy WM_TIMER tímto parametrem může být například identifikátor časovače, který zprávu vyvolal
d) parametr lParam – jedná se čtyřbytové číslo, které dále upřesňuje význam zprávy
e) čas kdy zpráva vznikla
f) pozice kurzoru myši v momentě, kdy zpráva vznikla

V zásadě existují dva způsoby, jakým může operační systém předat zprávu. Prvním způsobem je předat zprávu přímo funkci, která jí obslouží. Druhou možností je vložení zprávy do systémové fronty zpráv.

3.4 Handle

V operačním systému Windows je třeba, aby každý objekt bylo možné jednoznačně identifikovat (například z důvodu, že danému objektu bude třeba předat data). K tomu

slouží tzv. handly. Jedná se o objekty, které jsou vnitřně nejčastěji reprezentované jako ukazatele nebo jako identifikační čísla do tabulky spravovaných objektů.

3.5 Okna

Okna představují v prostředí Windows hlavní prvek grafického rozhraní. Bylo by však mylné domnívat se, že oknem je pouze hlavní okno aplikace. Za okno se ve Windows považuje každý grafický prvek, tedy i různé tlačítka, lišty, editační pole a další prvky, tedy jakékoliv instance třídy oken. Většina funkcí, které nám umožňují okna vytvářet, je umístěna v dynamicky linkovaných knihovnách kernel32.dll a user32.dll, které jsou součástí systému Windows.

Okna tvoří hierarchickou strukturu, ve které mohou (ale nemusí) mít svého rodiče či potomka. Představme si například hlavní okno, na které umístíme jiné okno, které bude představovat prvek *label*, tedy bude mít za úkol obsahovat pouhý text. Hlavní okno nebude mít žádného rodiče, ale bude mít potomka, již zmíněné okno představující *label*. Situace z pohledu vloženého okna bude přesně opačná. Vztah rodič – potomek je pro nás důležitý v souvislosti se zánikem oken, zanikne-li rodičovské okno, zaniknou s ním i všichni jeho potomci na něm umístění.

3.6 Knihovny Dll

V předchozí kapitole jsem zmínil některé dll knihovny a myslím, že by bylo na místě se s nimi seznámit. Dynamicky linkované knihovny (dynamic-link libraries), někdy zkráceně též nazývané dynamické knihovny, slouží k ukládání funkcí a procedur v podobě strojového kódu do samostatného souboru.

Mezi největší výhodu těchto knihoven patří fakt, že mnoho různých programů může využívat jedinou instanci knihovny a volat si z ní všechny potřebné funkce. Budeme-li mít například na počítači spuštěno několik různých aplikací, které potřebují vykonat naprosto stejnou operaci (například vytvořit a vykreslit okno), mohou si jednoduše zavolat funkci, která požadavek obslouží, z jediné dynamické knihovny. Kód funkce se tedy nachází mimo aplikaci a do paměti se zavádí při jejím spuštění a to navíc pouze v případě, že tam dosud nebyl zaveden jinou aplikací. Bez využití dll knihoven by daná funkce musela být pevnou součástí každé aplikace a být s každou aplikací znovu a znovu zaváděna do paměti, což by vedlo k jejímu neefektivnímu využívání.

Další výhodou dll knihoven je možnost jejich aktualizace. Bude-li potřeba změnit jedinou funkci využívanou několika aplikacemi, stačí ji změnit v dané knihovně přičemž nebude třeba vůbec přepisovat zmíněné aplikace.

Obsah dynamických knihoven však není omezen pouze na nejrůznější funkce, naopak mohou obsahovat například i data nebo obrázky.

Z výše popsaných důvodů jsou dll knihovny v operačním systému Windows velmi hojně zastoupeny. Například většina API funkcí, které Windows nabízí, je uložena v dynamických knihovnách (kernel32.dll, user32.dll, gdi32.dll).

3.7 Windows API

Jak již bylo napsáno výše, každá aplikace v prostředí Windows musí svoje požadavky provádět skrze operační systém, jelikož nemá přímý přístup k hardwaru. Ve skutečnosti však nekomunikují s jádrem Windows přímo, ale prostřednictvím služeb Windows API (application programming interface). Zjednodušeně řečeno, API (myšleno Win32 API, což je správný název) obsahuje všechny funkce a definice datových typů a souvisejících struktur, které může programátor využívat a lze s jeho pomocí provádět vše, co operační systém umožňuje.

V současné době existuje mnoho různých knihoven, které můžeme považovat za jakousi nadstavbu API, jmenujme například knihovny VCL (Visual Component Library od Borlandu) nebo MFC (Microsoft Foundation Class). Tyto knihovny, které může programátor ve svých aplikacích používat, obsahují mnoho užitečných funkcí a tříd, které udělají mnoho práce za něj. Nabízí se tedy otázka, proč vůbec využívat čisté API. Odpověď na tuto otázku bude poněkud nepřímá. Velkou nevýhodou výše zmíněných knihoven je, že mnohdy dokáží udělat mnohem více, než programátor potřebuje a ten pak kvůli drobnosti zavádí do paměti počítače zbytečné množství kódu, čímž zbytečně svoji aplikaci zpomaluje a navíc zabírá systémové prostředky počítače. Naproti tomu, jak říká Radek Chalupa [12], pomocí *API lze napsat malý, relativně jednoduchý prográmek bez výrazné ztráty produktivity a na druhou stranu s trvalým ziskem jeho efektivnosti, především z hlediska minimalizace nároků na paměť a další systémové zdroje*. Týká se to především programů běžících (většinou na pozadí) po celou dobu běhu operačního systému. To je také důvod, proč programování prostřednictvím API využívám ve své aplikaci.

4 Architektura programu s grafickým rozhraním

Každý program s grafickým rozhraním musí v prostředí Windows obsahovat čtyři základní stavební kameny. Jsou jimi funkce *WinMain(),smyčka zpráv*, *registrace třídy okna* a *procedura okna*.

4.1 Funkce WinMain

Funkce *WinMain()* představuje tzv. vstupní bod aplikace. Tato funkce je zároveň i funkcí hlavního vlákna programu. *WinMain()* obvykle obsahuje funkce sloužící k inicializaci programu (což zahrnuje i registraci třídy okna) a vytvoření hlavního okna. Zároveň obsahuje i smyčku zpráv. Následuje ukázka kódu:

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst, LPSTR
lpCmdLine, int nShow){
   MSG msg;
   if ( !InitApp() )  return FALSE;
   g_hInstance = hInstance;
   while ( GetMessage(&msg, NULL, 0, 0) )  {
     TranslateMessage(&msg);
     DispatchMessage(&msg);
  }
 return msg.wParam;
}
```

4.2 Registrace třídy okna

Jak je zmíněno v kapitole 3.5, každý grafický prvek je v systému Windows považován za okno. Není proto divu, že jednotlivé druhy oken jsou založeny na třídách, které definují vlastnosti jednotlivých oken.

Vytvoření okna není na první pohled úplně triviální procedura. Abychom mohli okno programu zobrazit, je nejdříve nutné zaregistrovat třídu okna. K tomu slouží funkce *RegisterClassEx()*. Nejprve je třeba nadefinovat třídu okna, což se děje pomocí naplnění struktury *WNDCLASSEX*, která je argumentem zmíněné funkce. Jedná se o poměrně rozsáhlou strukturu s mnoha parametry (obsahuje například informace o způsobu překreslování a stylu okna, handlu instance, handlu ikony, barvy pozadí), nebudu ji na tomto místě dopodrobna rozebírat. Při definici a následné registraci třídy okna můžeme mimo jiné změnit i některé parametry třídy a ovlivnit tak vlastnosti vytvářeného okna. Nutno podotknout, že není potřeba registrovat všechny třídy oken, které hodláme používat. Některé můžeme je rovnou používat, patří mezi ně zejména ty

systémové třídy, které reprezentují standardní prvky Windows jako tlačítka, edit-boxy apod. Příklad registrace třídy okna:

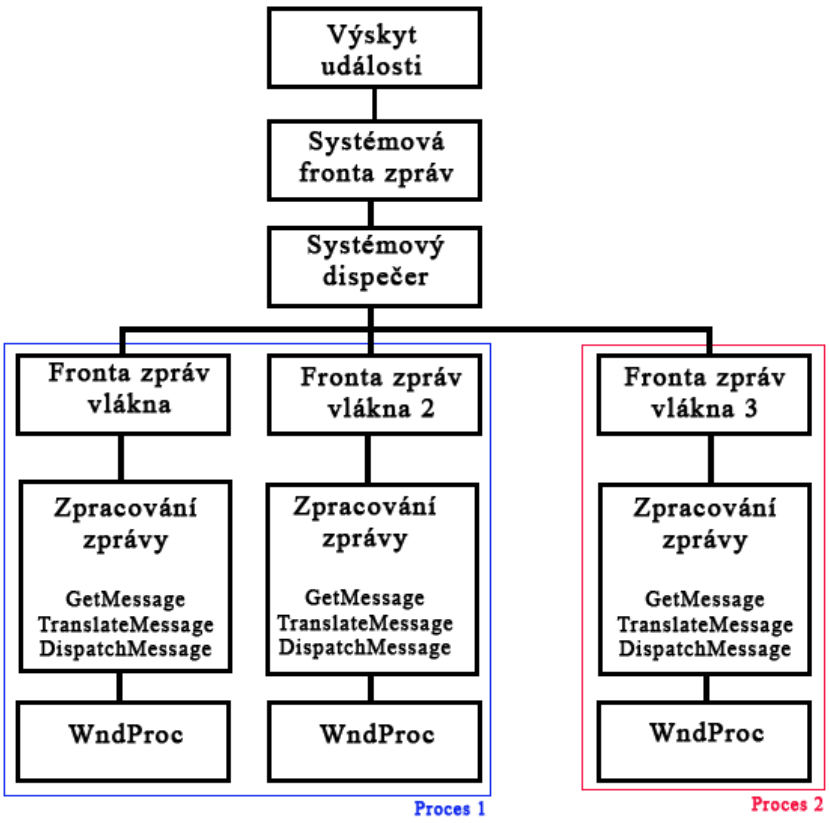
```
WNDCLASSEX wc;
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
   wc.hInstance = g_hInstance;
   wc.style = CS_HREDRAW | CS_VREDRAW;
   wcex.hIcon= LoadIcon(hInstance, MAKEINTRESOURCE(IDI_PLCCONTROLLER));
   wcex.hCursor= LoadCursor(NULL, IDC_ARROW);
   wcex.lpszClassName= szWindowClass;
  //nastavení dalších členů struktury
 RegisterClassEx(&wc)
```

4.3 Smyčka zpráv

Jak již bylo řečeno, operační systém vytváří zprávy při vzniku nejrůznějších událostí. Vytvořená zpráva je následně uložena do systémové fronty zpráv. Systémový dispečer pak jednotlivé zprávy přiřazuje jednotlivým vláknům. Každé vlákno má svoji vlastní frontu zpráv, ve které zpráva čeká do té doby, dokud není vyzvednutá funkcí *GetMessage()*. Odebírání zpráv pomocí této funkce se děje v nekonečné smyčce cyklu while (nekonečná smyčka bude přerušena, přijde-li zpráva WM_QUIT), následně je v tomtéž cyklu zpráva předzpracována pomocí funkce *TranslateMessage()* a předána funkci *DispatchMessage()*, která následně posílá data zprávy do procedury okna *WndProc*. Vyjádřeno zdrojovým kódem, situace vypadá následujícím způsobem.

```
while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }
```

Problematiku smyčky zpráv zachycuje následující obrázek



obrázek 2: Fronta zpráv

5 Procedura okna

Proceduře okna se předávají zprávy ze smyčky zpráv. V proceduře okna se nachází často velmi rozsáhlý příkaz switch, pomocí kterého zachytáváme jednotlivé druhy zpráv, na které chceme nějak reagovat. Samozřejmě druhů zpráv existuje velké množství a nemusíme reagovat na vše, co se proceduře okna předá. Jednoduchá procedura okna, která by sice příliš funkční nebyla, jelikož by reagovala pouze na zprávu k ukončení chodu aplikace, by mohla vypadat takto:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM
wParam, LPARAM lParam){
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

6 Popis rozhraní RS-232

Před vlastním rozborem programování přenosů dat po sériové lince v systému Windows by bylo vhodné seznámit se samotným rozhraním RS-232.

Poslední varianta standartu RS-232C vznikla v roce 1969 a používá se jako rozhraní ke komunikaci počítačů a další elektroniky. Umožňuje propojení mezi dvěma body. Data, respektive jednotlivé bity jsou posílány za sebou, neboli v sérii.

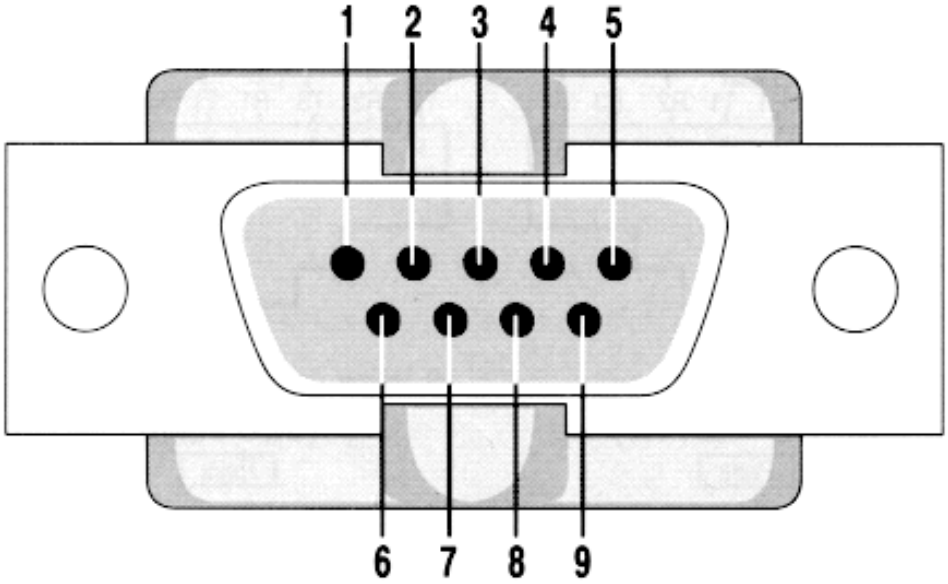
V současné době se sériová linka na osobních počítačích používá jen velmi málo. Důvodem je, že toto rozhraní je již značně zastaralé a je vytlačováno novými výkonnějšími rozhraními, například Universálním sériovým rozhraním USB (Universal Serial Bus). V oblasti průmyslu se však RS-232 či její modifikace RS-485 stále hojně využívají a dá se předpokládat, že tomu tak bude i nadále.

Dle tohoto standartu je možné komunikovat až na vzdálenost 15 metrů, nebo délku vodiče o kapacitě 2500 pF. To znamená, že při použití kvalitních vodičů lze dodržet standard a při zachování jmenovité kapacity prodloužit vzdálenost až na cca 50 metrů. Vodič lze také prodlužovat za předpokladu snížení přenosové rychlosti, protože potom bude přenos odolnější vůči velké kapacitě vedení. Výše uvedené parametry počítají s přenosovou rychlostí 19200 Bd. Narazil jsem na jednotku Baud, bude dobré se s ní také seznámit. Baud je jednotka používaná pro měření rychlosti modulace dat. Tato rychlost definuje rychlost přenosu dat z datového média na jiné datové médium. Baud rate udává počet změn signálu za sekundu. Jako základní jednotka informace v moderních počítačových systémech se bere jeden bit (nabývá hodnoty 0 nebo 1). Do jedné signálové změny lze zakódovat i více než jeden bit. A proto nelze slučovat pojem bps (bits per second = bity za sekundu) s pojmem baud. V případě některých typů modulací však může platit, že 1 baud je roven jednomu bitu.

Na obvyklých sériových portech v PC je možné dosáhnout rychlosti maximálně 115200 Bd. Často se však používají i rychlosti menší, tyto baudové rychlosti jsou odvozeny dělením 115200 Bd. Jedná se tedy například o rychlosti 57600 Bd, 38400 Bd, 28800 Bd, 23040 Bd, 19200 Bd, ..., 2400 Bd. Skutečná přenosová rychlost je však obvykle nižší než baudová rychlost a to z důvodu, že se s vlastními datovými bity přenášejí navíc startbity, paritní bity a stopbity.

Samotný port je obvykle realizovaný deviti-pinovým nebo pětadvaceti-pinovým konektorem D-sub. Na osobních počítačích se obvykle setkáme s devíti-pinovou variantou. Vzhledem k tomu, že toto rozhraní mělo umožňovat komunikaci s

modemem, nachází se na konektoru piny, které byly navrženy k odesílání a přijímání režijních signálů. Následující obrázek a tabulka vysvětlují význam jednotlivých pinů.



obrázek 3: Schéma konektoru sériového portu. Zdrojem obrázku je web AggSoftware [13]

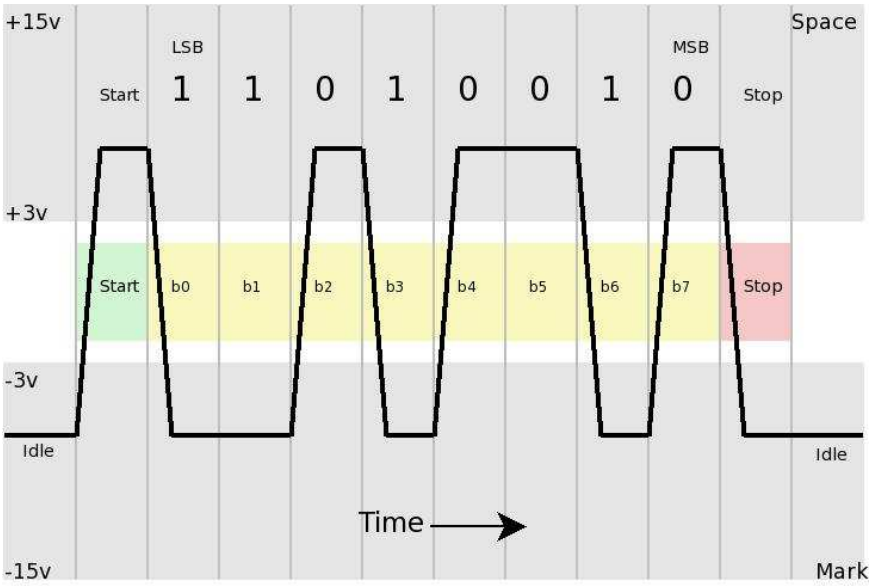
tabulka 1: Piny sériového portu

Číslo pinu	Signál	Význam signálu
1	CD	Carrier Detect – signalizuje terminálu, že byl na lince zachycen nosný signál
2	RxD	Receive Data – Tok dat z modemu do terminálu
3	TxD	Transmit Data – Tok dat z terminálu do modemu
4	DTR	Data Terminal Ready – Terminál signalizuje svoji připravenost komunikovat
5	GND	Ground – signálová zem
6	DSR	Data Set Ready – Modem signalizuje svoji připravenost komunikovat
7	RTS	Request to Send – Terminál signalizuje, že komunikační cesta je volná
8	CTS	Clear To Send – Modem signalizuje, že komunikační cesta je volná
9	RI	Ring Indicator - Modem signalizuje, že na telefonní lince detekoval signál vyzvánění

RS-232 umožňuje data odesílat asynchronním způsobem. To v praxi znamená, že se data přenáší data v určitých sekvencích. Z toho důvodu je nutné, aby oba účastníci komunikace měli stejná nastavení pro přenos dat. Informace jsou pak přenášeny přesně danou rychlostí a uvozeny startovacími bity, pomocí kterých se synchronizují všechna přijímací zařízení. Všechny strany obsahují vlastní přesný oscilátor, díky kterému odečítají data v přesně definovaných intervalech. Po ukončení sekvence je další příjem opět synchronizován startovacími bity.

Samotná sekvence začíná start bitem, po kterém zpravidla následuje 8 datových bitů (jejich skutečný počet může být ale i jiný), nepovinný paritní bit a zvolný počet stop bitů Nejčastěji využívaná varianta obsahuje celkem 10 bitů na sekvenci, 1 start bit, 8 datových bitů, 0 paritních, 1 stop bit. Samotná data v sekvenci bývají přenášena od nejméně významného bitu (LSB) po nejvíce významnému bitu (MSB).

RS-232 využívá dvou napěťových úrovní představujících logickou 0 a 1. Tyto hodnoty bývají většinou realizované pomocí napětí ± 5 V, ± 10V, ± 12V a ± 15V přičemž záporné hodnoty obvykle representují logickou 1 (též nazývána marking state) a kladné hodnoty logickou 0 (space state). Obrázek 6 zachycuje, jak by mohl vypadat signál, jímž by se přenášela binární hodnota 1010010.



obrázek 4: Průběh signálu reprezentující přenos binární hodnoty 11010010. Zdrojem obrázku je Wikipedia [8]

6.1 Parita

Parita představuje kontrolní mechanismus, kterým můžeme bez velkých systémových nároků ověřit, zda-li data dorazila v pořádku. Je-li použití parity při přenosu používáno, sečte se počet jedničkových bitů a data se doplní bitem paritním tak, aby platila podmínka sudého nebo lichého počtu jedničkových bitů. Přijímač pak má za úkol zkontrolovat, zda-li počet jedniček odpovídá nastavenému typu parity. Na paritu se však nelze příliš spoléhat, některé věci nedokáže odhalit. Například pokud bychom odeslali binární data 11110000 a přijali 10101010, parita by je vyhodnotila jako správně přijatá data ačkoliv jsou data na první pohled různá. Existují čtyři typy parity:

a) sudá – počet jedničkových bitů + bit paritní musí být sudé číslo

b) lichá – počet jedničkových bitů + bit paritní musí být liché číslo

c) nulová – paritní bit je vždy reprezentován nulou

d) jedničková – paritní bit je vždy reprezentován jedničkou

6.2 Řízení toku dat

Řízení toku dat neboli handshaking slouží k určení, zda-li komunikující zařízení jsou připravena přijímat či odesílat data. V praxi existují dvě různá řešení, hardwarové, které využívá piny RTS, CTS, DTR, DSR a softwarové, které je obvykle implementováno pomocí komunikačního protokolu. Použití řízení toku dat na RS-232 není povinné.

7 Programování sériového přenosu

Sériový port se z hlediska programování v operačním systému Windows považuje za soubor, tudíž se používají i stejné funkce jako při práci se soubory. Těmi nejdůležitějšími funkcemi jsou *CreateFile(),* která slouží k otevření portu, dále *ReadFile()* a *WriteFile(),* jejichž prostřednictvím z portu čteme a zapisujeme do něj. Nakonec funkcí *CloseHandle()* port opět uvolníme.

Nebude zřejmě překvapením, že port je třeba nejprve otevřít (či obsadit) a až poté jej můžeme používat k zápisu či čtení, nakonec bychom jej také měli uzavřít (uvolnit). Vzhledem k tomu, že sériový port je přeci jen speciálním typem souboru, je ještě nutné provést jeho konfiguraci, kterou nastavíme parametry přenosu. Vstupně/výstupní operace jako například zápis a čtení lze na Windows v zásadě

implementovat dvěma odlišnými přístupy, buď jako non-overlapped (nepřekrývané) nebo overlapped (překrývané) operace.

7.1 Non-overlapped operace

Používání *non-overlapped* přístupu je z programátorského hlediska jednodušší, avšak méně flexibilní. Pokud v našem programu budeme volat non-overlapped operaci, bude vlákno programu zablokováno do té doby, dokud operace neskončí. Zároveň ani žádné jiné vlákno, které by chtělo přistupovat k portu, nebude moci pokračovat, dokud se operace z prvního vlákna nevyřídí. Znamená to tedy, že v jednu chvíli může probíhat pouze jedna vstupně/výstupní operace.

7.2 Overlapped operace

Na rozdíl od předchozí možnosti je využívání *overlapped* operací výhodnější pro uživatele, avšak je náročnější na programování. Overlapped operace totiž umožňují, aby současně probíhalo více vstupně/výstupních operací a zároveň ani nezablokují vlákno během jejich vykonávání. Při tomto způsobu je však ale nutné kontrolovat, jestli se daná operace dokončila. Ta se totiž může dokončit ihned, ale také může skončit chybou ERROR_IO_PENDING, která signalizuje, že operace čeká na dokončení. V případě, že se operace dokončí, nastaví se ve struktuře typu OVERLAPPED událost, která vypovídá o tom, že byla operace dokončena. Reference na tuto strukturu se předává jako parametr funkcím *ReadFile()* a *WriteFile().* Overlapped přístupu budu využívat i v mé aplikaci.

7.3 Otevření portu

Jak již bylo výše zmíněno, k otevření portu se používá funkce *CreateFile()*. Nejdříve si ukážeme, jak se funkce používá a následně si seznámíme jejími parametry.

```
HANDLE hCom:
hCom = CreateFile(TCOM, GENERIC_READ | GENERIC_WRITE, 0 ,0 ,
OPEN_EXISTING, FILE_FLAG_OVERLAPPED,0);
if(hCom==INVALID_HANDLE_VALUE){
        if(GetLastError()==ERROR_FILE_NOT_FOUND){//chyba, port neexistuje
}
        else {//jiná chyba při otevírání portu, port je již například obsazen jinou aplikací
};
```

Pokud se funkci *CreateFile()* podaří otevřít port, získáme na něj také handle. Prvním parametrem funkce je systémové jméno portu, který chceme otevřít (např. COM1), druhý parametr určuje, že budeme z portu chtít jak číst, tak do něj zapisovat. Následující dva parametry ovlivňují sdílení a zabezpečení a musí být pevně nastaveny na hodnotu 0, pátý parametr musí být pevně nastaven na hodnotu OPEN_EXISTING. Zajímavý je pro nás šestý parametr, který určuje, zda-li port otevíráme pro non-overlapped nebo overlapped operace. Poslední parametr musí být taktéž pevně nastaven na hodnotu 0.

7.4 Nastavení portu

Před zahájením komunikace je nutné port nakonfigurovat. Dále je nutné, aby obě komunikující zařízení používali stejná nastavení.

Než se seznámíme se samotným nastavováním parametrů přenosu, bude třeba povědět si něco o struktuře DCB (Device Control Block). Tato struktura obsahuje 28 položek, pomocí kterých se nastavují parametry sériového přenosu. Nemá smysl na tomto místě vyjmenovávat všechny položky. Nejdůležitějšími z nich jsou rychlost přenosu, druh parity, počet datových bitů a počet stop bitů. V praxi se obvykle načte původní nastavení struktury pomocí funkce *GetCommState()*, následně se v *DBC* struktuře upraví jen ty parametry struktury, které nás zajímají a tato struktura se opět předá jako argument funkci SetCommState(), která port nakonfiguruje. Následuje ukázka nastavení některých parametrů komunikace prostřednictvím *DCB* struktury.

```
HANDLE hCom;
DCB dcbSerialParams = {0};
dcbSerialParams.DCBlength=sizeof(dcbSerialParams);
GetCommState(hCom, &dcbSerialParams);
dcbSerialParams.BaudRate=19200;
dcbSerialParams.ByteSize=8;
dcbSerialParams.StopBits=1;
dcbSerialParams.Parity=2;
if(!SetCommState(hCom,&dcbSerialParams)){
     //nepodařilo se nakonfigurovat port
}
```

Kromě těchto nastavení je také třeba nastavit časové limity (timeouty), jejich špatné nastavení by mohlo způsobit velké problémy s komunikací. Podobně jako v předchozím případě, i časové limity jsou uchovávány ve speciální struktuře zvané

COMMTIMEOUTS. Ta obsahuje následující položky: *ReadIntervalTimeout*, *ReadTotalTimeoutMultiplier*, *ReadTotalTimeoutConstant*, *WriteTotalTimeoutMultiplier* a *WriteTotalTimeoutConstant*, jejich význam nebudu na tomto místě dopodrobna popisovat. Obecně nastavují čas v milisekundách, který mají *ReadFile()* a *WriteFile()* funkce na to, aby vykonaly svoji funkci. Pomocí funkce *GetCommTimeouts()* můžeme získat současné nastavení časových limitů, naopak funkce *SetCommTimeouts()* slouží k jejich nastavení. Vyjádřeno kódem, situace vypadá následovně

```
HANDLE hCom;
COMMTIMEOUTS timeouts={0};
timeouts.ReadIntervalTimeout=50;
timeouts.ReadTotalTimeoutConstant=50;
timeouts.ReadTotalTimeoutMultiplier=10;
timeouts.WriteTotalTimeoutConstant=10;
timeouts.WriteTotalTimeoutMultiplier=10;
if(!SetCommTimeouts(hCom, &timeouts)){
     //nepodařilo se nastavit časové limity
;}
```

7.5 Čtení a zápis dat

K čtení a zápisu slouží již uvedené funkce *ReadFile()* a *WriteFile()*. Ačkoliv samozřejmě obě dělají něco jiného, fungují podobným způsobem. O tom, že mají hodně společného svědčí i fakt, že jejich parametry jsou shodné. Prvním argumentem je handle otevřeného portu, druhým je ukazatel na proměnnou, do které se data zapisují (čtou), dalšími dvěma argumenty jsou počty bytů, které se mají zapsat (přečíst) a počty které byly skutečně zapsány (přečteny), posledním argumentem je ukazatel na OVERLAPPED strukturu (v případě, kdy používáme non-overlapped operace a s touto strukturou nepracujeme, je parametr null)

7.5.1 Zápis dat

Zápis dat pomocí non-overlapped operace je vcelku jednoduchý, stačí využít funkci *WriteFile()* s příslušnými argumenty. Následuje ukázka zápisu dat.

```
HANDLE hComm;
DWORD dwWritten=0;
if(!WriteFile(hComm,pBuffer,dwToWrite,&dwWritten,NULL)){
        //chyba při zápisu dat
}
if (dwToWrite!=dwWritten) {
        //byl překročen časový limit
}
```

Při zápisu dat na port pomocí *overlapped* operace je třeba nejprve vytvořit proměnnou typu *OVERLAPPED* abychom v ní mohli pomocí funkce *CreateEvent()* vytvořit událost, která bude sloužit k oznámení o dokončení operace. Po samotném zavolání funkce WriteFile() je třeba otestovat, jestli funkce *WriteFile()* proběhla bezchybně nebo zda-li došlo k chybě. V případě, že k žádné chybě nedošlo, byl zápis dat vyřízen okamžitě. V opačném případě se okamžitě zápis dat nepodařilo vyřídit. Pomocí funkce *GetLastError()* můžeme zjistit, k jaké chybě došlo. Pokud došlo k chybě *ERROR_IO_PENDING*, tak to znamená, že se operaci nepodařilo vyřídit okamžitě, ale zápis bude opožděn. Pomocí funkce *GetOverlappedResult()* můžeme i otestovat, zda-li se nakonec zápis podařil. Nutno podotknout, že zápis by byl považován za vyřízený i v případě, že by došlo k vypršení časového limitu. To můžeme jednoduše otestovat tak, že porovnáme jestli se počet bytů, které jsme nechali zapsat, rovná počtu bytů, které byly skutečně zapsány. Pokud nikoliv, došlo k timeoutu.

7.5.2 Čtení dat

Čtení dat z portu se v zásadě příliš neliší od zápisování. Je zde však nutné řešit několik problémů navíc. Jedním z nich je, že potřebuje nejprve zjistit, jestli nějaká data po sériové lince dorazila do vstupního bufferu. To se dá zjistit pomocí funkce *WaitCommEvent()*, která nám oznámí, pokud na portu dojde k události specifikované za pomoci masky. Masky se nastavují pomocí funkce *SetCommMask()* a s jejich pomocí můžeme nastavit, jaké události budou na portu monitorovány.

Druhým problémem je, že potřebujeme zjistit, kolik dat na port dorazilo. Pokud zachytíme událost oznamující příchod dat, tak se pomocí funkce *ClearCommError()* a struktury C*OMSTAT* můžeme zjistit počet bajtů ve vstupním bufferu, které se následně budeme načítat. Pro non-overlapped funkci čtení by kód mohl vypadat následovně:

```
HANDLE hComm;
COMSTAT comStat;
DWORD dwError, dwCommEvent;
if (!SetCommMask(hComm, EV_RXCHAR)){
        //příznak EV_RXCHAR indikuje příchozí data
        //chyba, pří nastavení masky
}
if (WaitCommEvent(hComm, &dwCommEvent, NULL)){
        ClearCommError(hCom, &dwError, &comStat)
        //v comStat.cbInQue nalezneme, kolik bytů je ve vstupním bufferu
        ReadFile(hComm, &chRead, comStat.cbInQue, &dwRead, NULL)}
else {
//nastala chyba při WaitCommEvent
}

if (comStat.cbInQue!=dwRead) {
        //byl překročen časový limit
}
```

Při overloapped operaci by čtení bylo složitější, jelikož funkce *WaitCommEvent()* při overloapped operaci funguje jinak nečeká na výskyt události, ale říká, jestli událost nastala nebo ne. Museli bychom vytvořit *OVERLAPPED* strukturu a v ní pomocí funkce *CreateEvent()* vytvořit objekt události, který bude signalizovat příchod dat. Při vlastním čekáním na data bychom uplatnili funkci WaitForSingleObject(). Dále by bylo třeba nastavit pomocí funkce *SetCommMask()* sledovaný příznak *EV_RXCHAR* (příchod dat na port) a pomocí funkce *WaitCommEvent()* ověřovat, jestli už došlo k události specifikované maskou. Pokud ano, změnila by v objektu události stav na signalizovaný a funkce *WaitForSingleObject()* by poznala, že došlo k události. Následně bychom se mohli pokusit data přečíst.

7.6 Uzavření portu

Po ukončení práce s portem je nutné opět port uvolnit. To se provádí prostřednictvím funkce *CloseHandle()*, jejímž jediným parametrem je handle uvolňovaného portu.

8 SoftPLC

Jak již bylo zmíněno, SoftPLC emuluje v systému Windows řídící systém Tecomat TC700. Jedna instalace emulátoru umožňuje souběžný chod až čtyř různých programů běžících na SoftPLC.

Společnost Teco a.s. dodává SoftPLC na trh ve dvou licencích. A to placené plné verzi a ve verzi Trial, která je zdarma, avšak umožňuje spustit jen jeden program, který je navíc omezen na čtyři hodiny běhu.

Kromě vlastního výkonného programu bývá k programu přiložen i konfigurační nástroj. Úkolem tohoto konfiguračního nástroje je jednak vytvořit registrační soubor, který uživatel při registraci programu odesílá společnosti Teco, ale hlavně dva konfigurační .ini soubory. První konfigurační soubor nese údaje o uživateli, druhý pak obsahuje údaje o nastavení projektu.

Aplikace, kterou mám za úkol vytvořit, je konkrétně určena pro verzi 4.1 SoftPLC. Nutno podotknout, že v době psaní této práce není tato verze dostupná volně ke stažení a nebyl k ní dodáván konfigurační nástroj, avšak měla by být plně funkční.

V mém případě, kdy k mé verzi konfigurační nástroj není k dispozici, je jedinou možností .ini soubory upravovat ručně v některém textovém editoru a samotný výkonný program spouštět pomocí dávkového souboru s koncovkou .bat, ve kterém se specifikují cesty k oběma .ini souborům. Obsah takového dávkového souboru obsahuje jen text a může vypadat například takto:

softPLC.exe /P-C:\Program Files (x86)\SoftPLC\SoftPRJ.ini /I-C:\Program Files (x86)\SoftPLC\SoftPLC.ini

obrázek 5: Okno výkonného programu SoftPLC a první záložka konfiguračního nástroje

9 Síť EPSNET

Síť EPSNET slouží pro komunikace mezi PLC automaty a dalšími zařízeními. Na této síti mohou pracovat dva druhy účastníků, nadřízené stanice, které řídí komunikaci a podřízené stanice, které odpovídají na dotazy od nadřízené stanice.
Síť je možné provozovat v jedné ze dvou možných základních konfigurací:
a) Monomaster - v síti se nachází jedna nadřízená stanice a několik podřízených
b) Multimaster - v síti se nachází několik nadřízených stanic a několik podřízených

V našem případě budeme používat konfiguraci *monomaster*. Jak již bylo uvedeno, v tomto nastavení síť obsahuje jednu nadřízenou stanici (master), která řídí provoz na síti, a jednu či více podřízených stanic (slave). Počet podřízených stanic je omezen zpravidla přenosovým médiem, maximálně jich však může být jich může být 126. Zde je třeba zajistit, aby připojené PLC i nadřízený systém měly každý jinou adresu přičemž není nutné, aby adresy tvořili spojitou řadu.

V nastavení monomaster můžeme pohodlně vyměňovat data mezi nadřízenou stanicí a libovolnou podřízenou stanicí. Pokud bychom potřebovali vyměňovat data mezi dvěma podřízenými stanicemi, museli bychom je předávat přes stanici nadřízenou.

Běží-li PLC v komunikačním režimu *PC*, je na všech komunikačních kanálech dostupný kompletní soubor služeb sítě EPSNET, jehož součástí jsou jak systémové služby, určené k programování a ladění systému, tak i veřejné služby, určené k výměně dat. Omezující podmínkou je, že systémové služby podporuje v daném časovém okamžiku pouze jeden komunikační kanál

9.1 Obecná struktura komunikačního protokolu sítě EPSNET

Sériová komunikace v síti EPSNET nastavení, při kterém se data odesílají ve tvaru 1 start bit, 8 datových bitu, bit sudé parity, 1 stop bit. Aby účastníci komunikace odesílaným zprávám rozuměli, je nutné dodržet určitou podobu zprávy.

9.1.1 Komunikace ve směru nadřízená stanice → podřízená stanice

a) zpráva bez datového pole

SD1	DA	SA	FC	FCS	ED

b) zpráva s datovým polem

SD2	LE	LER	SD2R	DA	SA	FC	DATA	FCS	ED

9.1.2 Komunikace ve směru podřízená stanice → nadřízená stanice

a) odpověď bez datového pole

SACK

nebo

SD1	SA	DA	FC	FCS	ED

b) odpověď s datovým polem

SD2	LE	LER	SD2R	DA	SA	FC	DATA	FCS	ED

SD1 - úvodní znak 1 (start delimiter 1), pevná hodnota $10
SD2 - úvodní znak 2 (start delimiter 2), pevná hodnota $68
SD4 - úvodní znak 4 (start delimiter 4), pevná hodnota $DC
LE - délka dat (length), počet bytu položek DA+SA+FC+DATA
LER - opakovaná délka dat (length repeat), stejná hodnota jako LE
SD2R - opakovaný úvodní znak 2 (start delimiter 2 repeat), pevná hodnota $68
DA - cílová adresa (destination address), hodnoty 0 až 126
SA - zdrojová adresa (source address), hodnoty 0 až 126
FC - řídící byte rámce (frame control byte)
DATA - vlastní datové tělo zprávy
FCS - kontrolní součet (frame check sum), bytový součet všech bytu položek
DA, SA, FC a DATA se zanedbáním vyšších rádu vzniklých přenosem
ED - koncový znak (end delimiter), pevná hodnota $16
SACK - krátké potvrzení (short acknowledge), pevná hodnota $E5

10 Implementace programu.

Při programování této aplikace, kterou jsem pojmenoval *PLC Controller*, bylo třeba vyřešit několik hlavních problémů. Vůbec prvním krokem bylo, udělat si představu, jakým způsobem bude program pracovat a jak ho bude uživatel ovládat.

Vzhledem k tomu, že program slouží k přenosu dat mezi emulátorem a reálným automatem a veškerá činnost probíhá mimo zrak uživatele, bylo vhodné program přizpůsobit tomu, aby běžel na pozadí a uživatel se po jeho spuštění o něj nemusel příliš

starat. Prvním problémem bylo vyřešit způsob, jakým uživatel bude zadávat parametry programu. Bylo nanejvýš vhodné, aby program znal konfiguraci už při svém startu a ta se nemusela zadávat při každém spuštění programu (přinejmenším při ovládání stejné úlohy). Proto jsem se rozhodl načítat údaje o konfiguraci z konfiguračního souboru.

Dále bylo potřeba vyřešit způsob komunikace mezi navrhovaným programem a SoftPLC. Dle zadání práce toto mělo provádět pomocí sdílení paměti SoftPLC. Jelikož se jedná o čtení paměti externího programu, což není běžným způsobem možné (systém Windows nedovolí přístup k paměti cizí aplikace), bylo třeba se obrátit na společnost Teco, která SoftPLC vyvíjí. Jelikož společnost veřejně neuvádí způsob, jak se ke sdílené paměti dá přistupovat, bylo třeba se obrátit přímo na její zaměstnance. Po jistých problémech se mi podařilo od zmíněné společnosti získat materiály nutné k vyřešení problému. Jednalo se o hlavičkové soubory (neobsahují výkonný kód, jen deklarace funkcí a proměnných) a soubory typu .obj (obsahují již přeložený kód), které bylo třeba přidat do mého projektu, abych byl schopný založit instance tříd, které se využívají k přístupu do sdílené paměti.

Dalším krokem bylo vytvořit funkce, které by mi umožnily komunikaci po sériové lince. Jednalo se zejména o funkce pro otevření portu, konfiguraci portu, čtení dat, vysílání dat a uzavření portu. Vzhledem k tomu, že se všechny funkce týkají stejného problému, rozhodl jsem se je zapouzdřit do třídy.

Kromě samotné implementace přenosu dat po sériové lince bylo potřeba také zajistit přijímání a odesílání dat v podobě, které bude PLC automat rozumět. Bylo tedy nutné nastudovat komunikační protokol automatu a vymyslet způsob, jak data vysílat a přijímat v pevně stanoveném formátu.

Vzhledem k tomu, že automatu vysílám požadavky na různé služby, bylo také potřeba zjistit, na jakou službu jsem od automatu obdržel odpověď, neboli identifikovat přijatá data. Po jejich identifikaci je třeba data zpracovat (předat do paměti SoftPLC) a odeslat nový požadavek.

Aby měl uživatel kontrolu, že vše pracuje jak má, rozhodl jsem se také měřit počet vyřízených požadavků za jednu sekundu a prodlevu mezi zpracováním dat na PC a prodlevu odpovědi automatu. To mi také umožňuje detekovat stav, kdy SoftPLC nebo reálný automat nekomunikují a pokusit se o znovunavázání spojení.

Zjednodušeně, program pracuje zhruba tímto způsobem:

1) Přečtení konfiguračního souboru a inicializace programu

2) Zavedení vlákna pro neustálé čtení dat

3) Vytvoření vlákna pro odeslání dat a odeslání (vlákno se po odeslání ukončí)

4) Čekání na odpověď od automatu a přečtení dat

5) Identifikace dat

6) Zpracování dat

7) Vytvoření vlákna pro odeslání dat a odeslání (vlákno se po odeslání ukončí)

8) Opakování bodu 4-7

10.1 Načtení konfiguračního souboru

Pro načítání konfigurační souboru jsem si vytvořil funkci, která přečte všechny položky souboru a uloží je do globálních proměnných. Přitom jsem využíval funkcí *GetPrivateProfileString()* pro čtení řetězce a *GetPrivateProfileInt()* při čtení číselných hodnot. Následuje ukázka části funkce (načítaných položek je ve skutečnosti víc)

```
void ReadIniFile(){
 GetPrivateProfileString (TEXT("Komunikace"), TEXT("COM"), TEXT(""), COM,
15, TEXT("./Config.ini"));
dwBaudrate = (DWORD) GetPrivateProfileInt(TEXT("Komunikace"),
TEXT("Rychlost"), -1, TEXT("./Config.ini"));
}
```

10.2 Přístup k zápisníku pomocí modulu sdílené paměti

Jak již bylo uvedeno v kapitole 2.2.2, je možné pomocí modulu sdílené paměti, který je umístěn v knihovně ShmDll.dll (v předchozích verzích programu se knihovna jmenovala ShmSrv.dll) přistupovat k zápisníku SoftPLC. Výše zmíněná knihovna obsahuje jak data samotného zápisníku, tak i třídu *ShmSrv*, jejíž metody slouží k přístupu k datům. Dále obsahuje třídu ApiSrv, která obsahuje systémové funkce nutné pro chod SoftPLC, ale to pro nás není příliš důležitá. Než objasním význam

používaných funkcí, měl bych ještě uvést, co se pro účely SoftPLC rozumí pod pojmy *socket* a *port*.

Socket – jak již bylo řečeno, na SoftPLC mohou v jednu chvíli běžet až čtyři různé programy. Každý z nich běží na svém vlastním socketu a ten si můžeme představit jako určitý přidělený paměťový prostor.

Port – ke každému socketu se váže deset portů, přičemž první tři jsou obsazené systémem. Port si můžeme představit jako vstupní místo, přes které aplikace přistupuje k socketu. Například systémová konzola SoftPLC typicky obsazuje port 2.

Nyní již zbývá objasnit význam metod třídy ShmSrv, kterou využíváme k přístupu k zápisníku:

ShmGetInfo() – Parametrem této funkce je ukazatel na ukazatel na strukturu, která nese informace například o verzi programu či počtu připojení k modulu. Tato funkce naplní strukturu daty, které následně můžeme číst.

PortSetName() – Tato funkce slouží k nastavení jména aplikace (a popisu její funkce), která se k sdílenému modulu připojuje. Pod tímto jménem se poté aplikace hlásí v systémové konzoli SoftPLC (viz obr. 6). Parametrem funkce je číslo socketu, číslo portu a dále text, který reprezentuje jméno aplikace a její funkci.

PortResName() – Tato funkce je opakem funkce předchozí a slouží k uvolnění jména portu, parametrem je číslo socketu a portu.

OpenNoteEx() – Tato funkce je jedna ze stěžejních, otevírá totiž zápisník a vrací na něj ukazatel. Jejími parametry jsou čísla socketu a portu a adresa paměti, do které se zapíše zmíněný ukazatel

CloseNoteEx() – Účelem této funkce je naopak zápisník zase uzavřít poté, co s ní mukončíme práci. Parametrem jsou čísla socketu a portu.

GetNoteDcsEx() – Tato funkce funkce vrací ukazatel na strukturu (deskriptor zápisníku), která obsahuje ukazatele do jednotlivých zón zápisníku. S jejich pomocí poté jsme schopni číst data zápisníku a také je měnit. Parametry jsou čísla socketu, portu a adresa paměti, do které se uloží ukazatel na uvedenou strukturu.

GetPlcSW() – Parametry této funkce jsou čísla socketu a adresa, na kterou se ukládá ukazatel na strukturu, která obsahuje stavové slovo SoftPLC.

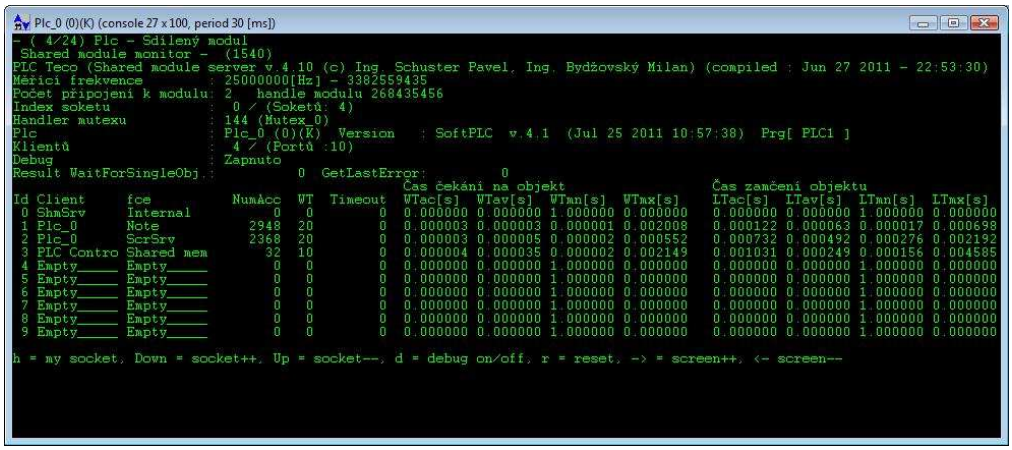
PlcLink() – Tato funkce ověřuje, zda-li je na daném socketu (parametr této funkce) skutečně běží program spuštění v SoftPLC.

ShmPortSetLock() – Tato funkce uzamkne zápisník, čímž se zajistí konzistence dat během jejich čtení či zápisu. Parametry jsou čísla socketu a portu a doba v milisekundách, na kterou se zápisník uzamkne.

PortResLock() – Tato funkce je velmi podobná té předchozí jen s tím rozdílem, že zápisník zase odemyká. I kdybychom tuto funkci nepoužili, SoftPLC by po době specifikované v jejím konfiguračním souboru samo převzalo kontrolu nad zápisníkem, ale spoléhat se na to a neuvolnit zápisník pomocí této funkce by bylo z programátorského hlediska neetické.

Následuje ukázka, jakým způsobem se přistupuje ke sdílené paměti:

```
Api = new ApiSrv();
Shm = new ShmSrv(Api, AnsiDllPath, 1, iShmResult);
if (!iShmResult){                       //test zavedení modulu sdílené paměti SoftPLC
     Shm->GetInfo(&pShmDll_Info);
     //nastavení jména služby (viditelné v konzoly SoftPLC)
     Shm->PortSetName(iSocket, iPort, "PLC Controller", "Shared memmory service");
    if (!Shm->OpenNoteEx(iSocket, iPort, &pz)){          //otevření zápisníku
        //není spuštěno SoftPLC;}
        //získání desktiptoru zápisníku
    Shm->GetNoteDscEx(iSocket, iPort, &dz);
      if (Shm->PlcLink(iSocket)) {                    //je-li navázáno spojení
          Shm->PortSetLock(iSocket, iPort, 10)        //uzamčení zápisníku
          //práce s daty, např.
          *(dz->Y+1)++;
          Shm->PortResLock(iSocket, iPort);    //ovolnění zápisníku
      }
      else{
          //není spojení se sdílenou pamětí
      }
Shm->CloseNoteEx(iSocket, iPort);           //uzavření zápisníku
Shm->PortResName(iSocket, iPort);           //uvolnění jména portu
if (Shm) delete Shm; //odstranění instancí tříd
if (Api) delete Api;
```



obrázek 6: Systémové konzola SoftPLC

10.3 Realizace přenosu po sériové lince

Nebudu na tomto místě podrobně rozebírat jak fungují funkce, které jsem napsal pro přenos dat po sériové lince, jelikož jsem podobný problém popisoval již v kapitole sedm. Stručně řečeno, nejprve dojde k otevření portu (ten je otevírán v OVERLAPPED režimu), následuje nastavení parametrů přenosu. Čtení dat obstarává vlákno, které je zavedeno v paměti po celou dobu běhu programu. Pro odesílání dat se spawnuje vždy nové vlákno, které po odeslání dat svou činnost ukončí.

10.4 Identifikace dat

Pro identifikaci dat jsem si napsal vlastní funkci. Vzhledem k tomu, že pro komunikaci s automatem využívám jen tři funkce, musím se starat o identifikaci čtyř možných odpovědí (čtvrtou odpovědí je obecná chyba, kterou automat může dávat najevo, že požadovanou službu nezná nebo není aktivní, případně, že nedokáže interpretovat přijatá data).

Identifikace dat je volána po každém úspěšném čtení dat z portu. Je-li bytů méně než šest, což je nejkratší odpověď, kterou na své požadavky mohu dostat, je funkce opuštěna a vrací hodnotu false (nepovedlo se identifikovat data). Je-li bytů šest či více, dochází k porovnávání bytů StartDelimiteru a FrameControlu, jejichž pomocí mohu odpovědi rozlišit. V případě odpovědi s datovým polem je mezi prvními šesti byty i byte, který říká, jak jsou data odpovědi dlouhá, takže je možné dopočítat celkovou délku odpovědi automatu. Při úspěšné identifikaci dat funkce vrací true a pomocí ukazatelů v argumentu funkce předávám délku odpovědi a číslo, kterým odpověď identifikuji pro další zpracování.

11 Závěr

Práce je věnována návrhu a realizaci softwaru, který umožňuje řídit fyzikální modely připojené k reálnému PLC automatu za pomoci emulátoru SoftPLC. Výsledný program umožňuje načítat data z registrů automatu přes sériovou linku a předávat je přímo do zápisníku SoftPLC, které data podle uživatelského programu zpracuje. Tyto zpracovaná data jsou následně opět odesílána PLC automatu. Touto výměnou dat je realizováno samotné řízení.

Výsledný program byl naprogramován v jazyce C++ a je určen pro platformu Windows a pro svůj chod vyžaduje verzi Windows 2000 a novější. Zároveň je určen k řízení pomocí SoftPLC verze 4.1. Tento program je přiložen k práci.

Základem pro úspěšné vypracování bylo seznámit se s možnostmi práce s modulem sdílené paměti, který je situován v dynamické knihovně ShmDll.dll, jež je součástí instalace SoftPLC. Dále bylo třeba seznámit se programováním pomocí funkcí Windows API a nastudovat informace o sériovém rozhraní RS-232, respektive nastudovat, jakým způsobem se programuje sériová komunikace v prostředí Windows. V neposlední řadě bylo nutné seznámit se se sítí EPSNET a jejím komunikačním protokolem.

V mojí práci je také nastíněno, jakým způsobem pracují aplikace na systému Windows a proč je výhodné programování pomocí čistého Windows API. Dále byly čtenáři přiblíženy možnosti emulátoru SoftPLC, z nichž i plyne, proč bychom se rozvojem těchto virtuálních PLC měli zabývat do budoucna.

Funkce programu byla ověřena pomocí připojení k reálnému automatu řady Tecomat TC600. Během testování bylo podle očekávání pozorováno, že při přenosu menšího počtu datových bytů roste při stejné rychlosti komunikace počet vyřízených požadavků (výměn dat mezi emulátorem a automatem) za sekundu.

Dá se předpokládat, že se pro své výhody stane řízení pomocí SoftPLC v budoucnosti značně populární. Z toho důvodu by možná do budoucnosti nebylo na škodu stávající program rozšířit o další funkce, například mu umožnit řídit více automatů najednou nebo využívat pro řízení i jiné komunikační kanály než jen sériovou linku.

Seznam použité literatury a zdrojů

[1] ECKEL, Bruce: Myslíme v jazyku C++. Praha: Grada Publishing, 2000. ISBN 80-247-9009-2

[2] PRATA, Stephan: Mistrovství v C++. 2. vyd. Brno: Computer Press, 2004. ISBN 80-251-0098-7

[3] VACEK, Václav: Sériová komunikace ve Win32, BEN – technická literatura, 2003. ISBN 80-7300-086-5

[4] LIBERTY, Jesse, Naučte se C++ za 21 dní, Computer Press, 2002, ISBN 80-7226-744-4

[5] Teco a.s., Sériová komunikace programovatelných automatů tecomat - model 32 bitů [online]. březen 2009. dostupné z WWW: < http://www.tecomat.cz/docs/cze/General/txv00403.pdf>

[6] Teco a.s., Příručka SoftPLC for Windows [online]. září 2007. dostupné z WWW: < http://www.tecomat.cz/docs/cze/Tecomat/txv13862.pdf>

[7] OLMR, Vít. HW server představuje - Sériová linka RS-232 [online]. 12.12.2005, dostupné z WWW < http://hw.cz/rs-232>

[8] Wikipedia. RS-232 [online]. červenec 2011. dostupné z WWW: < http://cs.wikipedia.org/wiki/RS-232>

[9] DENVER, Allen. Serial Communications in Win32 [online]. 11.12.1995. Dostupné z WWW:

<http://msdn2.microsoft.com/en-us/library/ms810467.aspx>.

[10] KLEIN, Ramon de. Serial library for C++ [online] 13.11.2003. Dostupné z WWW:

< http://www.codeproject.com/KB/system/serial.aspx>

[11] DHAR, Asnish. Serial Communication in Windows. 3.8. 2002. Dostupné z WWW: < http://www.codeproject.com/KB/system/serial_com.aspx>

[12] CHALUPA, Radek. Seriál Učíme se Win API [online]. 1.11. 2002. Dostupné z WWW: <http://www.builder.cz/art/cpp/winapi1.html>.

[13] AggSoftware. PC Com Port [online]. Dostupné z WWW: <http://www.aggsoft.com/rs232-pinout-cable/serial-cable-connections.htm>

Příloha A – Uživatelská příručka pro PLC Controller

Úvod

PLC Controller je aplikace určená pro řízení reálných PLC automatů Tecomat pomocí softwarového emulátoru SoftPLC for Windows (verze 4.1).

Řízení probíhá na principu výměny dat pomocí sériové linky RS-232 mezi registry reálného automatu a sdílenou pamětí SoftPLC běžícím na PC. Konkrétně jsou čteny vstupní registry X reálného automatu a jejich obrazy jsou zapsány na příslušná místa ve sdílené paměti SoftPLC, tudíž emulátor má k dispozici stejná vstupní data jako reálný automat. Emulátor pak může data zpracovat na základě svého programu a výstupy zapsat do výstupních registrů Y své sdílené paměti. Obrazy těchto výstupů jsou následně přečteny a přeneseny do výstupních registrů reálného automatu, takže i ten má k dispozici výstupní data zpracovaná emulátorem.

Dochází tedy k situaci, kdy reálný automat pouze obsluhuje fyzické vstupy a výstupy a za zpracování dat, respektive za řízení celého systému, zodpovídá SoftPLC.

Systémové požadavky

PLC Controller je možné používat na libovolném PC s nainstalovaným a aktualizovaným operačním systémem Microsoft Windows 2000 / XP / Vista / 7. Je tedy potřeba, aby příslušný počítač splňoval hardwarové nároky příslušného operačního systému. Vzhledem k povaze aplikace je také nutné, aby byl na počítači k dispozici sériový port COM.

Samotný PLC Controller pak vyžaduje zhruba 5 MB volného místa na pevném disku a obsadí zhruba 1,6 MB operační paměti.

K chodu aplikace je mimo jiné potřeba, aby na počítači byl nainstalován produkt *Microsoft Visual C++ 2010 Redistributable Package (x86),* obsahující komponenty nutné pro běh aplikací vyvinutých ve vývojovém prostředí Microsoft Visual Studio 2010. Zmíněný produkt je možné získat zdarma na webových stránkách Microsoftu, konkrétně na adrese

http://www.microsoft.com/download/en/details.aspx?id=5555

Instalace

PLC Controller není potřeba instalovat v pravém slova smyslu. Postačí jen nakopírovat hlavní spustitelný soubor PLC Controller.exe do zvoleného umístění. Vzhledem k tomu, že nedochází k modifikaci registrů Windows, není potřeba ani žádného speciálního postupu při odstraňování aplikace ze systému, opět postačí zmíněný soubor běžným způsobem smazat.

Součásti instalace

Jak již bylo výše zmíněno, vlastní aplikace se skládá z jediného spustitelného souboru PLC Controller.exe, který představuje vlastní aplikaci.

Kromě ní je však přikládán ještě nástroj PLC Controller-Configurator. Jedná je o jednoduchou formulářovou spustitelnou aplikaci, pomocí které uživatel nastavuje parametry PLC Controlleru, respektive vytváří konfigurační soubor, který nese údaje o nastavení a je načítán při každém spuštění programu.

Konfigurace programu

Aby program správně fungoval, je nutné předat mu informace o nastavení. Vzhledem k tomu, že PLC Controller je primárně určen pro běh na pozadí, není k dispozici způsob, jak změnit nastavení za běhu (navíc k tomu není ani praktický důvod). Místo toho se informace o nastavení předávají prostřednictvím konfiguračního souboru Config.ini, který je načítán při spuštění programu. Konfigurační soubor se musí nacházet ve stejné složce jako hlavní spustitelný soubor programu. K vytváření konfiguračního souboru slouží výše zmíněný nástroj PLC Controller-Configurator. Význam nastavovaných parametrů je následující:

Cesta k souboru ShmDll.dll – Knihovna ShmDll.dll je součástí instalace SoftPLC for Windows verze 4.1 a obsahuje jak samotnou sdílenou pamět SoftPLC, tak i funkce, které program využívá k přístupu k ní. Proto je nutné znát a zadat adresu umístění této knihovny a to v absolutním tvaru (například. C:\Program files\SoftPLC\ShmDll.dll)

Socket – SoftPLC v registrované plné verzi umožňuje běh až čtyř různých programů najednou, s trial licencí pouze jednoho. Socketem se dle vývojářů SoftPLC označuje číslo instance spuštěného programu. Jinými slovy tento parametr definuje, ke kterému

programu (respektive sdílené paměti) se budeme připojovat. Tento parametr tedy nabývá hodnot 0 až 3. Používáme-li trial licenci, bude tento parametr vždy nula.

Port – K sdílené paměti se v rámci jednoho socketu může současně připojovat více než jen jedna aplikace. Port představuje číslo, pod kterým se daná aplikace ke sdílené paměti připojuje. Pro každý socket je k dispozici dohromady deset portů, přičemž první tři jsou rezervovány pro potřeby SoftPLC. Uživatel se tedy může připojit na kterýkoliv z portů 3 až 9 (za předpokladu, že se k nim nepřipojuje nějaká další aplikace). Po úspěšném připojení aplikace k modulu sdílené paměti můžeme v systémové konzoly SoftPLC vidět, že se PLC Controller připojil k danému portu.

Rychlost – Jak již název napovídá, toto nastavení určuje, pro jakou rychlost komunikace v baudech za sekundu bude nakonfigurován sériový port v počítači. Je nutné, aby stejnou rychlostí komunikoval i reálný PLC automat. Rychlost není libovolná, je možné využít jen předdefinované hodnoty.

Parita - Tento parametr určuje, zda-li se při přenosu dat pomocí sériové linky bude používat parita a jakého bude druhu. Reálný automat musí používat stejný druh parity.

Počet stop bitů - Toto nastavení určuje, kolik stop bitů má sériový port vysílat (a očekávat) v každém datovém rámci. Pro úspěšný přenos reálný automat musí být nakonfigurován stejně. Pozor při manuálním editování konfiguračního souboru. Chceme-li používat 1 stopbit, zadává se hodnota 0, při 1,5 stopbitu hodnota 1 a při 2 stopbitech hodnota 2.

Jméno portu – Tímto nastavíme, na kterém sériovém portu bude aplikace komunikovat. Jedná se jméno portu, pod kterým se port hlásí v operačním systému, například COM1.

Začátek čtení – Tento parametr slouží k určení bloku čtených dat z reálného automatu, respektive určuje, který registr se bude číst jako první. Například budeme-li začínat číst data od registru X0, hodnotou tohoto parametru je 0. Bude-li začínat číst od registru X5, hodnotou parametru je 5.

Kolik číst – Tímto se nastavuje počet čtených registrů X. Například budeme-li chtít číst registry X0 až X5, nastavíme parametr jako 6. K specifikování čteného bloku dat je tedy potřeba nastavit začátek čtení a kolik bytů číst.

Začátek zápisu – Obdoba začátku čtení, slouží k určení začátku bloku zapisovaných dat z registru Y emulátoru.

Kolik zapsat – Tímto parametrem určujeme, kolik dat výstupních dat z registru Y se bude zapisovat.

SoftPLC for Windows verze 4.1

PLC Controller využívá pro připojení ke sdílené paměti SoftPLC některé zdrojové soubory poskytnuté přímo vývojářem SoftPLC, společností Teco. Jejich použití bylo nezbytné a zároveň i předurčilo i kompatibilitu programu. Pouze SoftPLC verze 4.1 je kompatibilní s PLC Controllerem.

Mohu se jen domnívat, že tato verze zřejmě zůstala jen pro interní využití firmy Teco, jisté je, že oficiálně nebyla dosud (prosinec 2011) uvolněna pro veřejnost. A ačkoliv by měla být plně funkční, její použití není zdaleka tak jednoduché, jako u předchozích verzí. Domnívám se, že by bylo na místě rozepsat detaily či rozdíly.

Instalace SoftPLC for Windows 4.1

Společnost Teco předchozí verze distribuovala v podobě instalačního souboru SetupSoftPLC.exe, pomocí kterého se SoftPLC nainstalovalo na počítač i příslušným nástrojem pro import licence a nastavení. Tato verze se neinstaluje, postačí ji nakopírovat do zvoleného umístění. V případě potřeby ji můžeme jednoduchým způsobem opět smazat. Bohužel, zmíněný nástroj chybí…

Import licence SoftPLC for Windows 4.1

Import licence je zřejmě největším kamenem úrazu této verze. V předchozích verzích se pomocí konfiguračního nástroje, který sloužil zároveň i jako průvodce importem licence, vytvořil soubor s informacemi o uživateli a s hardwarovým otiskem počítače. Tento soubor se následně přes webové rozhraní odeslal společnosti Teco a ta zpětně pomocí e-mailu poslala uživateli soubor představující licenční klíč, pomocí kterého se importovala licence.

Bohužel, SoftPLC 4.1 nemá k dispozici zmíněný nástroj pro konfiguraci a import licence. Není tedy způsob, jakým jednoduše vygenerovat zmíněný soubor s informacemi o uživateli k odeslání společnosti Teco a tudíž jak importovat licenci.

Jediný způsob, jaký je mi znám a jakým se problém dá obejít, vede přes instalaci starší verze SoftPLC (úspěšně testováno s verzí 3.3). Je potřeba nainstalovat starší verzi, zaregistrovat a importovat licenci (postup je popsán v Příručce pro SoftPLC for Windows dostupné na webových stánkách společnosti Teco).

Po importu licence bychom měli mít k dispozici dva textové soubory pojmenované *SoftPLC_Info.TXT* a *SoftPLC_Info_key.TXT*. Nepříjemné je, že tyto

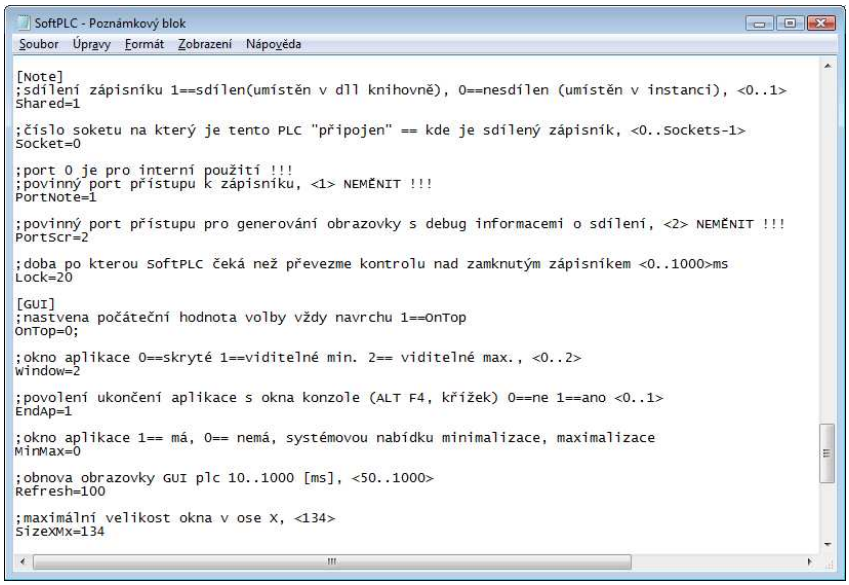
soubory nenajdeme v umístění, kam jsme instalovali SoftPLC. Ve skutečnosti si SoftPLC zakládá další svojí složku v dokumentech a její absolutní umístění se bude lišit v závislosti na operačním systému. Například v mém případě se na 64b verzi operačního systému Windows Vista tyto soubory nachází v umístění *C:\Users\All Users\SoftPLC* přičemž složka *All Users* je ještě ke všemu běžně nastavena jako skrytá.

Povede-li se nám zmíněné soubory vypátrat, stačí je jen oba nakopírovat do složky, v do které jsme SoftPLC verze 4.1 umístili.

Konfigurace SoftPLC for Windows 4.1

Absence konfiguračního nástroje přináší ještě jeden podstatný problém a tím je uživatelsky nepříliš příjemná možnost nastavení.

Informace o uživateli jsou ukládány v konfiguračním souboru SoftPRJ.ini uloženém ve složce se SoftPLC. Důležitější systémové konfigurační údaje jsou ukládány v konfiguračním souboru SoftPLC.ini, který se nachází v téže složce. Naštěstí je možné tyto konfigurační soubory poměrně jednoduše manuálně editovat prostřednictvím jakéhokoliv textového editoru, použít můžeme například Poznámkový blok, který je běžnou součástí Windows. Útěchou nám také může být, že jednotlivé položky souboru jsou relativně srozumitelně pojmenovány a nechybí ani komentáře s významem a množinou přípustných parametrů, takže konfigurace s trochou zkušeností možná je. Za názvem parametru a rovnítkem se uvádí hodnota parametru.



obrázek 7: Ukázka obsahu konfiguračního souboru SoftPLC.ini

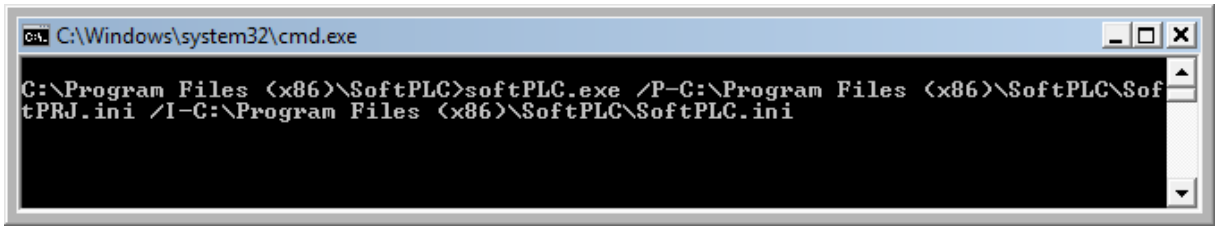
Spuštění SoftPLC for Windows 4.1

Samotné spuštění této verze SoftPLC se také liší od předchozích verzí, kde se nejdříve pomocí konfiguračního nástroje zadala konfigurace a následně se pomocí tlačítka *Spustit* spustil i samotný program.

SoftPLC v této verzi se spouští pomocí dávkového souboru *RunPLC.bat*. Dávkové soubory jsou ve skutečnosti běžnými textovými soubory, jejichž obsahem jsou příkazy, které jsou předávány interpretu *Příkazového řádku*. V našem případě zmíněný dávkový soubor obsahuje příkaz, který spouští SoftPLC a zároveň jako parametr příkazu jsou předány absolutní cesty ke konfiguračním souborům SoftPLC.ini a SoftPRJ.ini. Bude tedy nutné nejprve dávkový soubor editovat a správně nastavit absolutní cesty ke konfiguračním souborům. K editaci můžeme opět použít libovolný textový editor, například Poznámkový blok. V případě, že se SoftPLC nachází v umístění C:\Program Files (x86)\SoftPLC\, příkaz v dávkovém souboru by vypadal následovně:

```
softPLC.exe /P-C:\Program Files (x86)\SoftPLC\SoftPRJ.ini
/I-C:\Program Files (x86)\SoftPLC\SoftPLC.ini
```

Po spuštění dávkového souboru dojde ke zpracování příkazu interpretem Příkazového řádku a pokud jsme udělali vše správně, dojde i ke spuštění samotného SoftPLC.



obrázek 8: Ukázka příkazového řádku při spouštění SoftPLC pomocí dávkového souboru

Uživatelské rozhraní PLC Controlleru

PLC Controller je od samého počátku zamýšlen pro běh na pozadí operačního systému. Většina úkonů, které program vykonává, se děje mimo zrak uživatele, není ani potřeba měnit parametry programu za jeho běhu. Z těchto důvodů uživatelské rozhraní obsahuje minimum ovládacích prvků.

Pomocí tlačítka pro minimalizaci okna se program laicky řečeno nepřesune běžným způsobem do hlavního panelu, kde by mohl překážet, ale je skryt do

oznamovací oblasti (system traye), kde se následně zobrazí jeho ikonka. Dvojklikem na ní je možné okno programu obnovit.

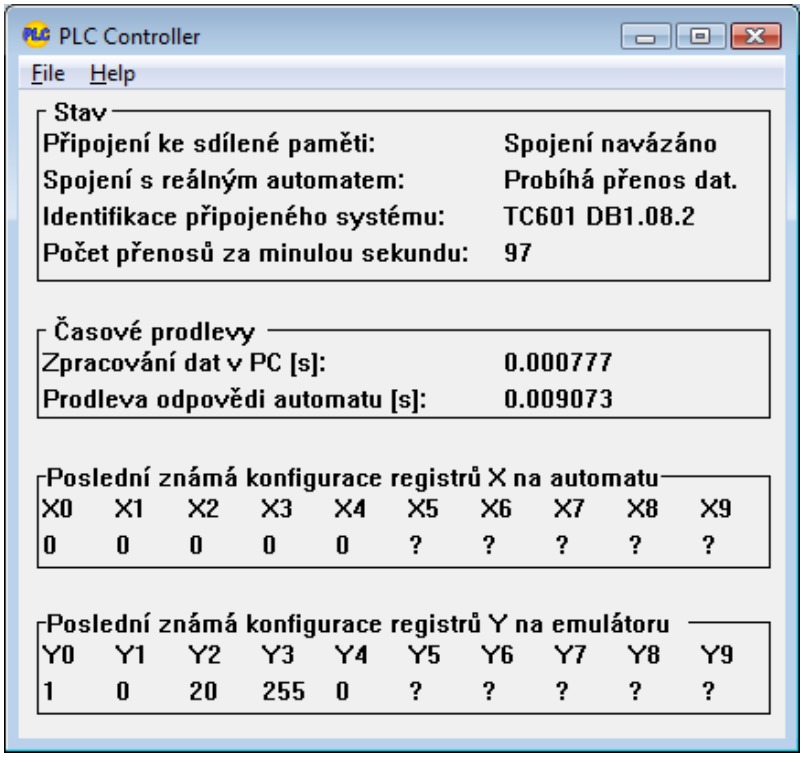
Aby uživatel měl možnost ověřit si, že program pracuje, program informuje uživatele prostřednictvím výpisů v okně o aktuálním stavu. Konkrétně jsou zobrazovány informace o stavu připojení ke sdílené paměti SoftPLC a zda-li probíhá aktivní komunikace mezi programem a reálným automatem.

V případě úspěšného navázání komunikace s automatem je rovněž zjišťována informace o druhu připojeného systému, tedy identifikační řetězec centrální jednotky.

Dále je monitorován počet úspěšně vyřízených požadavků na přenos dat mezi reálným automatem a SoftPLC za předchozí sekundu.

Mimo to je sledována doba, která uplynula mezi přijetím prvního bytu odpovědi od automatu, zpracováním dat a odesláním nového požadavku na data do automatu (aneb celková doba zpracovávání dat v PC). Následně je měřena i doba mezi odesláním požadavku na data a přijetím prvního bytu odpovědi (aneb prodleva mezi odesláním požadavku na data a přijetím odpovědi). Tyto funkce pro měření prodlev nemusí být vždy dostupné, zejména na starších počítačích často chybí jejich hardwarová podpora.

V poslední řadě je pro kontrolní účely vypisován i poslední známý stav vstupních registrů X automatu a výstupních registrů Y emulátoru.



obrázek 9: Okno PLC Controlleru